Using the latest UICollectionView APIs

Intro

— Joshua Kaplan (`yhkaplan`)
— Senior iOS Engineer @ GMO Pepabo working on minne
— Tooling, frameworks, and architecture
— Stay-at-home shiba-inu parent
— He/him

Topics

— Layout
— Data and Cell Configuration
— Future Directions

Topics skipped

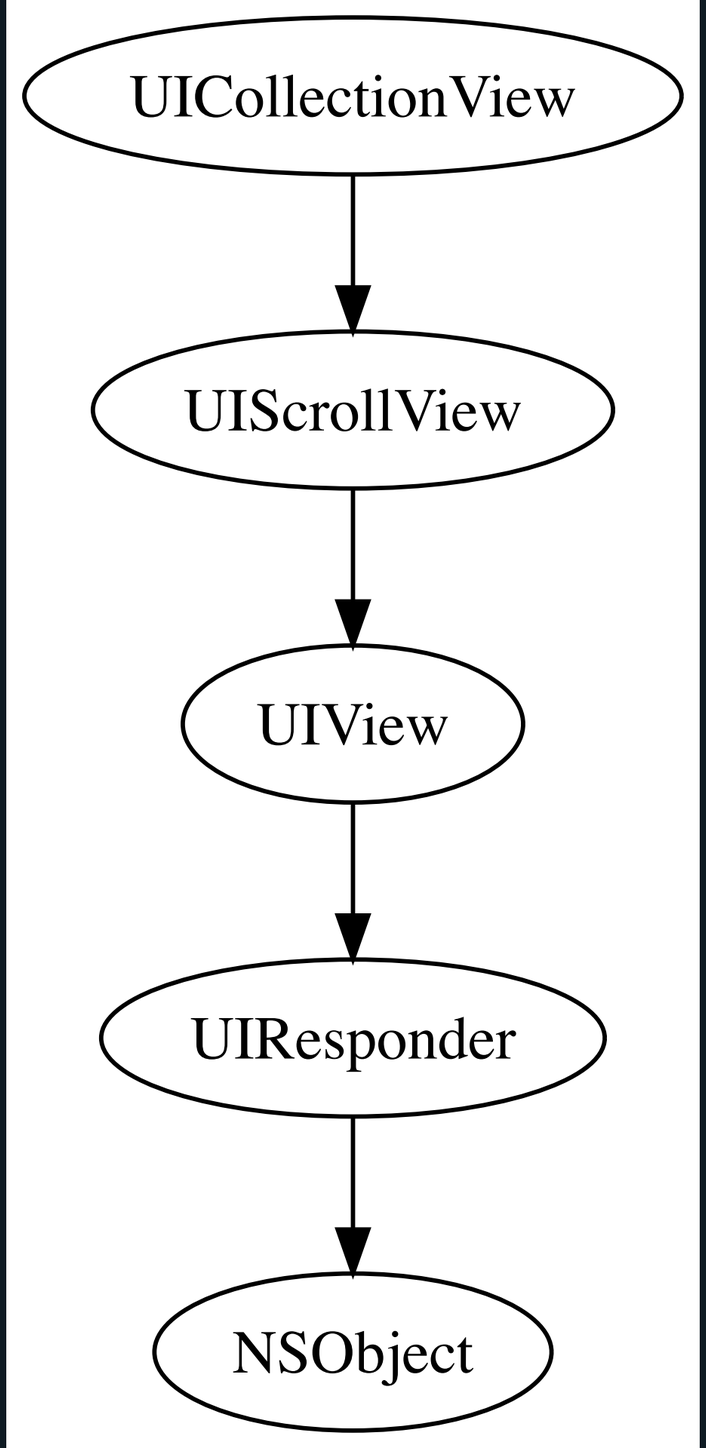
— Drag and drop
— Cell editing
— Headers and footers
— Prefetching
— Dynamic layout transitions

Layout

Definition

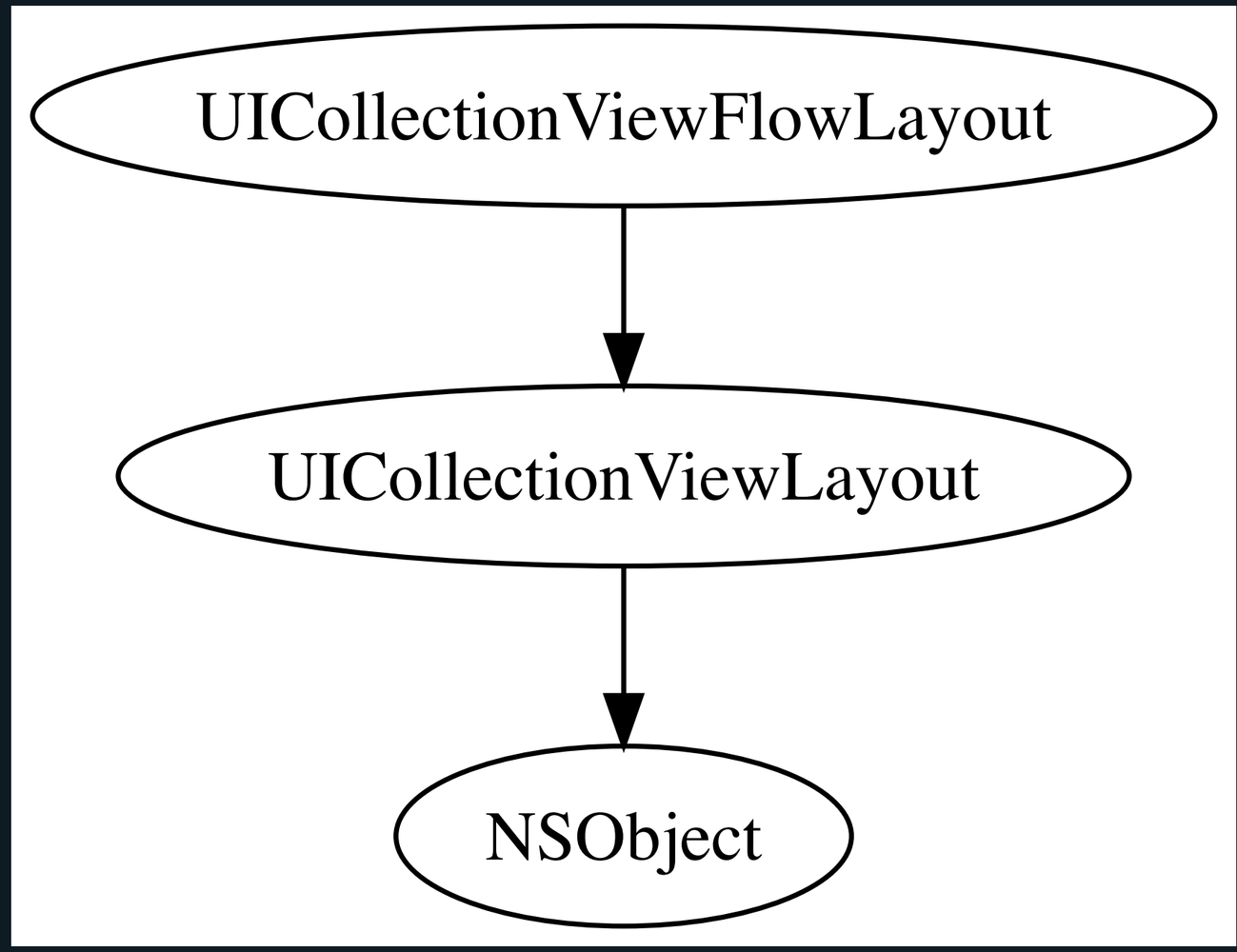
- Standard grid, complex grid, list, and more
- Difference w/ UITableView
- Manages multiple scrolling views
  - **completely configurable** layout
  - High performance, view recycling

Class hierarchy



Basic Grid

- UICollectionViewFlowLayout
- Define in code UICollectionViewFlowLayout or delegate, or interface builder



```swift
final class CollectionViewBasicsVC: UIViewController {
    private lazy var data = (0...100).compactMap { _ in
        ["pencil", "trash", "paperplane", "calendar", "lightbulb"].randomElement()
    }
    private lazy var layout: UICollectionViewFlowLayout = {
        let l = UICollectionViewFlowLayout()
        let halfWidth = view.bounds.width / 2
        let halfWidthMinusMargins = halfWidth - 14
        let height = halfWidthMinusMargins
        l.itemSize = CGSize(width: halfWidthMinusMargins, height: height)
        return l
    }()
    private lazy var collectionView = UICollectionView(frame: view.bounds, collectionViewLayout: layout)


    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(collectionView)

        collectionView.contentInset = UIEdgeInsets(top: 8, left: 8, bottom: 8, right: 8)
        collectionView.backgroundColor = .white
        collectionView.register(BasicCell.self, forCellWithReuseIdentifier: BasicCell.reuseID)
        collectionView.dataSource = self
    }
}

extension CollectionViewBasicsVC: UICollectionViewDataSource {
    func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        data.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: BasicCell.reuseID, for: indexPath)

        let systemName = data[indexPath.item]
        if let image = UIImage(systemName: systemName) {
            (cell as? BasicCell)?.configure(with: image)
        }
        return cell
    }
}
```

Complex Grid

— UICollectionView CompositionalLayout

— Complex, grouped sections

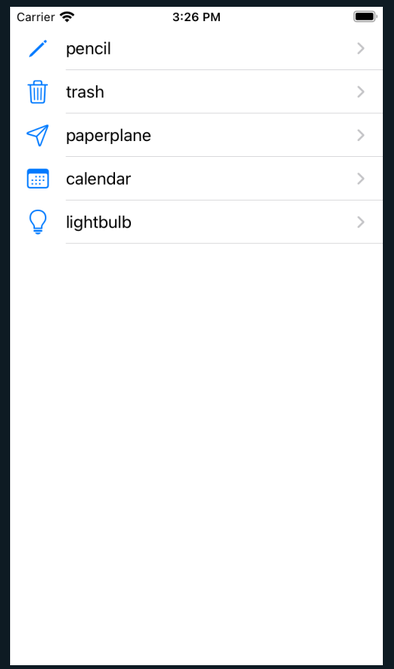
— Convenient for future proofing simpler views

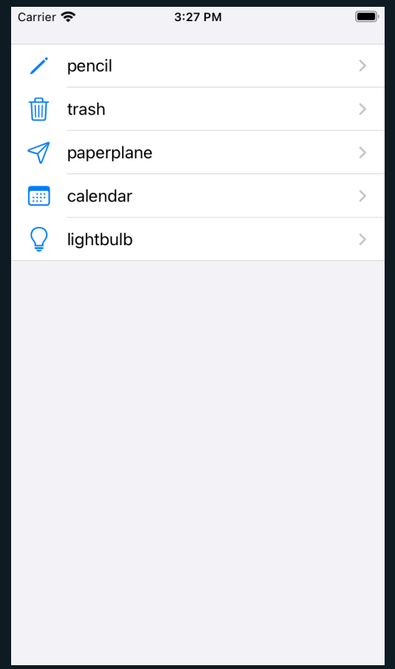
Code

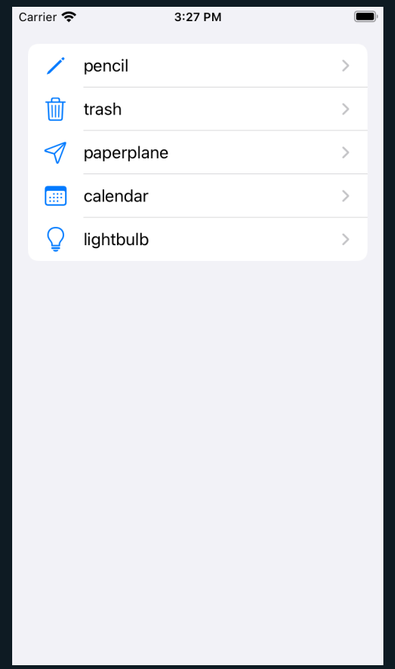
```
final class BasicCompositionalLayoutGridVC: UIViewController {
    enum Section: Hashable { case grid }

    private let data = ["pencil", "trash", "paperplane", "calendar", "lightbulb"]

    private lazy var collectionView = UICollectionView(frame: view.bounds, collectionViewLayout: layout)
    private lazy var layout = UICollectionViewCompositionalLayout { sectionIndex, environment in
        let itemSize = NSCollectionLayoutSize(widthDimension: .fractionalWidth(0.5), heightDimension: .fractionalHeight(1.0))
        let item = NSCollectionLayoutItem(layoutSize: itemSize)
        item.contentInsets = NSDirectionalEdgeInsets(top: 4, leading: 4, bottom: 4, trailing: 4)

        let groupSize = NSCollectionLayoutSize(widthDimension: .fractionalWidth(1.0), heightDimension: .fractionalWidth(0.5))
        let group = NSCollectionLayoutGroup.horizontal(layoutSize: groupSize, subitems: [item])

        return NSCollectionLayoutSection(group: group)
    }
    private lazy var dataSource = UICollectionViewDiffableDataSource<Section, String>(collectionView: collectionView) {...}

    override func viewDidLoad() {...}
}
```

List

— iOS 14+
— Part of CompositionalLayout
— All main UITableView styles available







```swift
final class ListVC: UIViewController {
    enum Section: Hashable { case list }

    private let data = ["pencil", "trash", "paperplane", "calendar", "lightbulb"]

    private lazy var collectionView = UICollectionView(frame: view.bounds, collectionViewLayout: layout)
    private lazy var layout: UICollectionViewCompositionalLayout = {
        let config = UICollectionLayoutListConfiguration(appearance: .plain)
        return UICollectionViewCompositionalLayout.list(using: config)
    }()
    private lazy var dataSource = UICollectionViewDiffableDataSource<Section, String>(
        collectionView: collectionView
    ) { collectionView, indexPath, item in
        let registration = UICollectionView.CellRegistration<UICollectionViewListCell, String> { cell, indexPath, item in
            var content = cell.defaultContentConfiguration()
            content.image = UIImage(systemName: item)
            content.text = item
            cell.contentConfiguration = content
        }

        let cell = collectionView.dequeueConfiguredReusableCell(using: registration, for: indexPath, item: item)
        cell.accessories = [.disclosureIndicator()]

        return cell
    }

    override func viewDidLoad() {...}
}
```

Completely configurable🤔

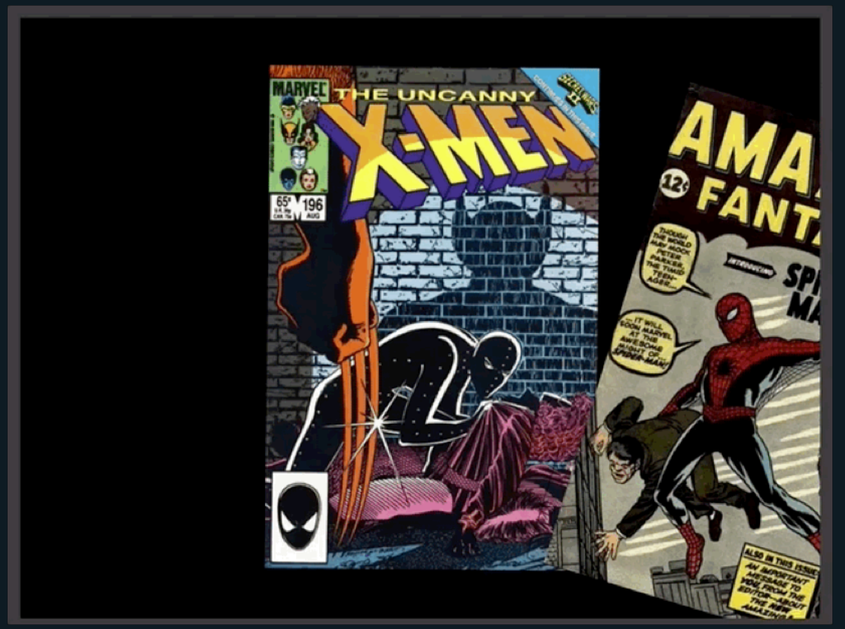
— Not just grids and lists!

— 3D stacks, carousels, or anything

Examples

Spinner

— jVirus/uicollectionview-layouts-kit

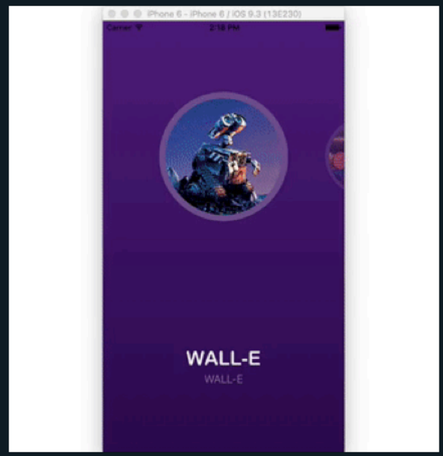


Safari

— jVirus/uicollectionview-layouts-kit

Carousel

— zepojo/UPCarouselFlowLayout

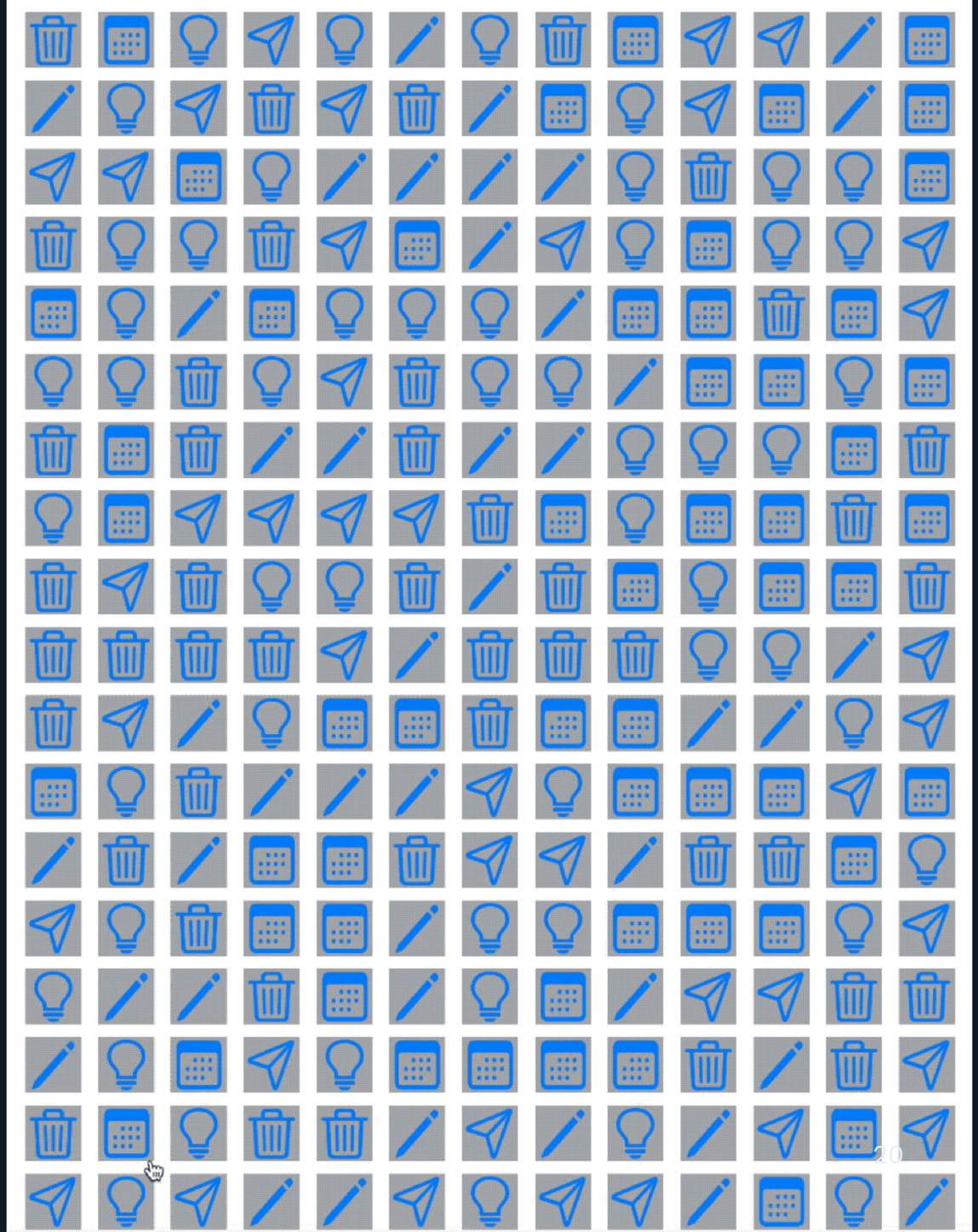


BounceyLayout

— GitHub - roberthein/BouncyLayout: Make. It. Bounce.

Custom Layout

- Subclass UICollectionViewFlowLayout or UICollectionViewLayout
- Procedural and verbose
- Powerful
- UIKitDynamics
- Easy to pop-in OSS layouts



```swift
open class BouncyLayout: UICollectionViewFlowLayout {
    lazy var dynamicAnimator = UIDynamicAnimator(collectionViewLayout: self)
    var latestDelta: CGFloat = 0.0
    var visibleIndexPaths: Set<IndexPath> = []

    override init() {...}
    required public init?(coder: NSCoder) {...}

    override open func prepare() {
        super.prepare()

        // Need to overflow our actual visible rect slightly to avoid flickering.
        guard let collectionView = collectionView else { return }

        let rect = CGRect(origin: collectionView.bounds.origin, size: collectionView.frame.size)
        let visibleRect = rect.insetBy(dx: -100.0, dy: -100.0)
        guard let itemsInVisibleRect = super.layoutAttributesForElements(in: visibleRect) else { return }
        let itemsIndexPathsInVisibleRect: Set<IndexPath> = Set(itemsInVisibleRect.map { $0.indexPath })

        // Step 1: Remove any behaviors that are no longer visible.
        let noLongerVisibleBehaviors = dynamicAnimator.behaviors.filter { behavior in
            guard
                let behaviorItem = (behavior as? UIAttachmentBehavior)?.items.first,
                let layoutAttribute = behaviorItem as? UICollectionViewLayoutAttributes
            else { return false }

            return !itemsIndexPathsInVisibleRect.contains(layoutAttribute.indexPath)
        }

        noLongerVisibleBehaviors.forEach { behavior in
            dynamicAnimator.removeBehavior(behavior)
            if let layoutAttribute = (behavior as? UIAttachmentBehavior)?.items.first as? UICollectionViewLayoutAttributes {
                visibleIndexPaths.remove(layoutAttribute.indexPath)
            }
        }

        // Step 2: Add any newly visible behaviors.
        // A "newly visible" item is one that is in the itemsInVisibleRect(Set|Array) but not in the visibleIndexPathsSet
        let newlyVisibleItems = itemsInVisibleRect.filter { !visibleIndexPaths.contains($0.indexPath) }
        let touchLocation = collectionView.panGestureRecognizer.location(in: collectionView)

        newlyVisibleItems.forEach { item in
            var center = item.center
            let springBehavior = UIAttachmentBehavior(item: item, attachedToAnchor: center)

            springBehavior.length = 0.0
            springBehavior.damping = 0.8
            springBehavior.length = 1.0

            // If our touchLocation is not (0,0), we'll need to adjust our item's center "in flight"
            if CGPoint.zero != touchLocation {
                let yDistanceFromTouch = abs(touchLocation.y - springBehavior.anchorPoint.y)
                let xDistanceFromTouch = abs(touchLocation.x - springBehavior.anchorPoint.x)
                let scrollResistance = (yDistanceFromTouch + xDistanceFromTouch) / 1_500.0

                if latestDelta < 0 {
                    center.y += max(latestDelta, latestDelta * scrollResistance)
                } else {
                    center.y += min(latestDelta, latestDelta * scrollResistance)
                }
                item.center = center
            }

            dynamicAnimator.addBehavior(springBehavior)
            visibleIndexPaths.insert(item.indexPath)
        }
    }

    open override func layoutAttributesForElements(in rect: CGRect) -> [UICollectionViewLayoutAttributes]? {
        return dynamicAnimator.items(in: rect) as? [UICollectionViewLayoutAttributes]
    }

    open override func layoutAttributesForItem(at indexPath: IndexPath) -> UICollectionViewLayoutAttributes? {
        return dynamicAnimator.layoutAttributesForCell(at: indexPath)
    }

    open override func shouldInvalidateLayout(forBoundsChange newBounds: CGRect) -> Bool {
        guard let collectionView = collectionView else { return false }
        let scrollView = collectionView

        let delta = newBounds.origin.y - scrollView.bounds.origin.y
        latestDelta = delta

        let touchLocation = collectionView.panGestureRecognizer.location(in: collectionView)

        dynamicAnimator.behaviors.forEach { behavior in
            guard let springBehavior = behavior as? UIAttachmentBehavior else { return }
            let yDistanceFromTouch = abs(touchLocation.y - springBehavior.anchorPoint.y)
            let xDistanceFromTouch = abs(touchLocation.x - springBehavior.anchorPoint.x)

            let scrollResistance = (yDistanceFromTouch + xDistanceFromTouch) / 1_500.0

            if let item = springBehavior.items.first as? UICollectionViewLayoutAttributes {
                var center = item.center
                if delta < 0 {
                    center.y += max(delta, delta * scrollResistance)
                } else {
                    center.y += min(delta, delta * scrollResistance)
                }
                item.center = center

                dynamicAnimator.updateItem(usingCurrentState: item)
            }
        }

        return false
    }
}
```

Data and Cell Configuration

UICollectionViewDiffableDataSource

- Fits most cases
- iOS 14 and 15 added
  - Cell/section reordering
  - Updating specific sections
  - Reloading completely w/o diff for better performance on large changes
- Animation behavior difficult to customize
- SE-0240: Ordered Collection Diffing is your friend
  - Swift 5.1
  - Find inserted, deleted, and updated items/sections, then simply use `performBatchUpdates(_:completion:)`

Code

```
final class BasicCompositionalLayoutGridVC: UIViewController {
    enum Section: Hashable { case grid }

    private let data = ["pencil", "trash", "paperplane", "calendar", "lightbulb"]

    private lazy var collectionView = UICollectionView(frame: view.bounds, collectionViewLayout: layout)
    private lazy var layout = UICollectionViewCompositionalLayout {...}
    private lazy var dataSource = UICollectionViewDiffableDataSource<Section, String>(collectionView: collectionView) { collectionView, indexPath, item in
        let registration = UICollectionView.CellRegistration<UICollectionViewCell, String> { cell, indexPath, item in
            let image = UIImage(systemName: item)
            cell.contentConfiguration = ImageContentView.Config(image: image)
        }

        return collectionView.dequeueConfiguredReusableCell(using: registration, for: indexPath, item: item)
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        view.addSubview(collectionView)

        var snapshot = NSDiffableDataSourceSnapshot<Section, String>()
        snapshot.appendSections([.grid])
        snapshot.appendItems(data, toSection: .grid)
        dataSource.apply(snapshot, animatingDifferences: false)
    }
}
```

Cell configuration and updating

— Useful for UITableView-style cells
— Custom cells maybe more work than preferable

List example

```
final class ListVC: UIViewController {
    enum Section: Hashable { case list }

    private let data = ["pencil", "trash", "paperplane", "calendar", "lightbulb"]

    private lazy var collectionView = UICollectionView(frame: view.bounds, collectionViewLayout: layout)
    private lazy var layout: UICollectionViewCompositionalLayout = {
        let config = UICollectionLayoutListConfiguration(appearance: .plain)
        return UICollectionViewCompositionalLayout.list(using: config)
    }()
    private lazy var dataSource = UICollectionViewDiffableDataSource<Section, String>(collectionView: collectionView) { collectionView, indexPath, item in
        let registration = UICollectionView.CellRegistration<UICollectionViewListCell, String> { cell, indexPath, item in
            var content = cell.defaultContentConfiguration()
            content.image = UIImage(systemName: item)
            content.text = item
            cell.contentConfiguration = content
        }

        let cell = collectionView.dequeueConfiguredReusableCell(using: registration, for: indexPath, item: item)
        cell.accessories = [.disclosureIndicator()]

        return cell
    }

    override func viewDidLoad() {...}
}
```

Custom example

UIContentView

```
final class ImageContentView: UIView, UIContentView {
    private var _configuration: Config
    private let imageView = UIImageView()

    var configuration: UIContentConfiguration {
        get { _configuration }
        set {
            guard let config = newValue as? Config else { return }
            _configuration = config
            imageView.image = _configuration.image
        }
    }

    init(config: Config) {
        _configuration = config
        super.init(frame: .zero)

        backgroundColor = .lightGray

        layoutImageView()
        imageView.image = _configuration.image
    }

    private func layoutImageView() {...}

    @available(*, unavailable)
    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    struct Config: UIContentConfiguration {
        let image: UIImage?

        func makeContentView() -> UIView & UIContentView { ImageContentView(config: self) }
        func updated(for state: UIConfigurationState) -> ImageContentView.Config { self }
    }
}
```

Registration

```
final class BasicCompositionalLayoutGridVC: UIViewController {
    enum Section: Hashable { case grid }

    private let data = ["pencil", "trash", "paperplane", "calendar", "lightbulb"]

    private lazy var collectionView = UICollectionView(frame: view.bounds, collectionViewLayout: layout)
    private lazy var layout = UICollectionViewCompositionalLayout {...}
    private lazy var dataSource = UICollectionViewDiffableDataSource<Section, String>(collectionView: collectionView) { collectionView, indexPath, item in
        let registration = UICollectionView.CellRegistration<UICollectionViewCell, String> { cell, indexPath, item in
            let image = UIImage(systemName: item)
            cell.contentConfiguration = ImageContentView.Config(image: image)
        }

        return collectionView.dequeueConfiguredReusableCell(using: registration, for: indexPath, item: item)
    }

    override func viewDidLoad() {...}
}
```

Updating

- Update based on UICellConfigurationState w/o subclassing cell
  - UICellConfigurationState: isSelected, isHighlighted, isDisabled, etc
- reconfigureItems(_:)
- configurationUpdateHandler: UICollectionViewCell.ConfigurationUpdateHandler?

```
cell.configurationUpdateHandler = { cell, state in
    var content = UIListContentConfiguration.cell().updated(for: state)
    content.text = "Hello world!"
    if state.isDisabled {
        content.textProperties.color = .systemGray
    }
    cell.contentConfiguration = content
}
```

Future Directions

— List and Grid

— Performance

— Customizability

```
struct HomeView: View {
    private let columns = [GridItem(.adaptive(minimum: 80)), GridItem(.adaptive(minimum: 80))]
    var body: some View {
        ScrollView {
            LazyVGrid(columns: columns) {
                ForEach(viewModel.products, id: \.id) { product in
                    ProductCard(product)
                }
            }
        }
    }
}
```

When to use UITableView

— complex list-style customization
— self-sizing cells w/ dynamic height are generally easier
— Consider using SwiftUI.List too

Conclusion

- Find the right fit
- Older ways and SwiftUI work too, but know what the latest APIs are
- You mostly don't need UITableView anymore
- UICollectionView is really flexible and performant
- SwiftUI's Grid views are still somewhat lacking in comparison

References and Miscellaneous

OSS Examples

- Various layouts
  - jVirus/uicollectionview-layouts-kit
  - amirdew/CollectionViewPagingLayout
- Abstracting many layouts WenchaoD/FSPagerView
- Neat slanted layout yacir/CollectionViewSlantedLayout
- Abstraction based on delegates airbnb/MagazineLayout
- Pinterest-like layout ChernyshenkoTaras/SquareFlowLayout
- Carousel layout: zepojo/UPCarouselFlowLayout
- UIKitDynamics GitHub - roberthein/BouncyLayout: Make. It. Bounce.

Timeline

- 2016 or earlier / iOS 10
  - DataSourcePrefetching
- 2017 or earlier / iOS 11
  - drag and drop
- 2019 / iOS 13
  - DiffableDataSource
  - CompositionalLayout
- 2020 / iOS 14
  - cell configuration
  - UICollectionView list style
  - DiffableDataSource & CompositionalLayout new features
- 2021 / iOS 14.5
  - UIListSeparatorConfiguration
- 2021 / iOS 15
  - cell configuration new features
  - DiffableDataSource & CompositionalLayout new features

Reference

— UICollectionViewLayout

— About iOS Collection Views

WWDC Videos

— Advances in UICollectionView
— Make blazing fast lists and collection views
— Advances in diffable data sources
— Lists in UICollectionView
— Modern cell configuration
— A Tour of UICollectionView
— Drag and Drop with Collection and Table View
— What's New in UICollectionView in iOS 10